Aggregation and Degradation in JetStream: Streaming Analytics in the Wide Area

Ariel Rabkin
Princeton University
asrabkin@cs.princeton.edu

Work done with Matvey Arye, Siddhartha Sen, Vivek S. Pai, and Michael J. Freedman
Today’s Analytics Architectures

- Backhaul is inefficient and inflexible
Tomorrow’s Architecture: JetStream

- Backhaul is inefficient and inflexible
- Goal: optimize use of WAN links by exposing them to streaming system.
Backhaul is Intrinsically Inefficient

- **Buyer’s remorse:** wasted bandwidth
- **Analyst’s remorse:** system overload or missing data

Graph showing the relationship between bandwidth and time over two days, with areas labeled "Needed for backhaul" and "Available".
Stream Processing Basics

Some Operators in JetStream:
- Filtering (count > 100)
- Sampling (drop 90% of data)
- Image Compression
- Quantiles (95th percentile)
- Query stored data
The JetStream System

**What:** Streaming with aggregation and degradation as first-class primitives

**Where:** Storage and processing at edge

**Why:** Maximize goodput using aggregation and degradation

**How:** Data cubes and feedback control
An Example Query

How popular is every URL?
Mechanism 1: Storage with Aggregation

Every minute, compute request counts by URL
Mechanism 2: Adaptive Degradation

Every minute, compute request counts by URL
Requirements for Storage Abstraction

- **Update-able** (locally and incrementally)

  ![Stored Data + Data](image)

- **Data size is reducible** (with predictable accuracy cost)

  ![Data → Data](image)

- **Merge-able** (without accuracy penalty)

  ![Data + Data = Merged Representation](image)
The Data Cube Model

Cube: A multidimensional array, indexed by a set of dimensions, whose cells hold aggregates.

<table>
<thead>
<tr>
<th>Counts by URL</th>
<th>12:00</th>
<th>12:01</th>
<th>12:02</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.mysite.com/a">www.mysite.com/a</a></td>
<td>3</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td><a href="http://www.mysite.com/b">www.mysite.com/b</a></td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td><a href="http://www.yoursite.com">www.yoursite.com</a></td>
<td>5</td>
<td>4</td>
<td>...</td>
</tr>
<tr>
<td><a href="http://www.her-site.com">www.her-site.com</a></td>
<td>8</td>
<td>12</td>
<td>...</td>
</tr>
</tbody>
</table>

Aggregation used for:
- Updates
- Roll-ups
- Merging cubes
- Summarizing cubes

Cubes have aggregation function: \( \text{Agg}(\text{cube}, \text{cube}) \rightarrow \text{cube} \)
Cubes can be “Rolled Up”

Cube: A multidimensional array, indexed by a set of *dimensions*, whose cells hold *aggregates*.

<table>
<thead>
<tr>
<th>Counts by URL</th>
<th>12:00</th>
<th>12:01</th>
<th>12:02</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.mysite.com/a">www.mysite.com/a</a></td>
<td>3</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td><a href="http://www.mysite.com/b">www.mysite.com/b</a></td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td><a href="http://www.yoursite.com">www.yoursite.com</a></td>
<td>5</td>
<td>4</td>
<td>…</td>
</tr>
<tr>
<td><a href="http://www.her-site.com">www.her-site.com</a></td>
<td>8</td>
<td>12</td>
<td>…</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Counts by URL</th>
<th>*</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.mysite.com/a">www.mysite.com/a</a></td>
<td>8</td>
</tr>
<tr>
<td><a href="http://www.mysite.com/b">www.mysite.com/b</a></td>
<td>2</td>
</tr>
<tr>
<td><a href="http://www.yoursite.com">www.yoursite.com</a></td>
<td>9</td>
</tr>
<tr>
<td><a href="http://www.her-site.com">www.her-site.com</a></td>
<td>20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Counts by URL</th>
<th>12:00</th>
<th>12:01</th>
<th>12:02</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>16</td>
<td>23</td>
<td>…</td>
</tr>
</tbody>
</table>
Cubes Unify Storage and Aggregation

Stored Data

Update

Standing Query

Update sent downstream

One-off query
Degradation: The Big Picture

- Level of degradation auto-tuned to match bandwidth.
- Challenge: Supporting mergeability and flexible policies
Mergeability Imposes Constraints

- Insight: Degradation may be discontinuous
There Are Many Ways to Degrade Data

- Can *coarsen* a dimension

- Can drop low-rank values
Coarsening Does Not Always Help

Savings from Aggregation

<table>
<thead>
<tr>
<th>Aggregation time period</th>
<th>Domains</th>
<th>URLs</th>
</tr>
</thead>
<tbody>
<tr>
<td>5s</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>minute</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5 m</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>hour</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>day</td>
<td>256</td>
<td>256</td>
</tr>
</tbody>
</table>
## Degradations Have Trade-offs

<table>
<thead>
<tr>
<th>Name</th>
<th>Fixed BW Savings</th>
<th>Fixed Accuracy cost</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dim. Coarsening</td>
<td>Usually no</td>
<td>Yes</td>
<td>Dimension Scale</td>
</tr>
<tr>
<td>Drop values (locally)</td>
<td>Yes</td>
<td>No</td>
<td>Cut-off</td>
</tr>
<tr>
<td>Drop values (globally)</td>
<td>No, multi-round protocol</td>
<td>Yes</td>
<td>Cut-off</td>
</tr>
<tr>
<td>Audiovisual downsampling</td>
<td>Yes</td>
<td>Yes</td>
<td>Sample rate</td>
</tr>
<tr>
<td>Histogram Coarsening</td>
<td>Yes</td>
<td>Yes</td>
<td>Number of Buckets</td>
</tr>
</tbody>
</table>
A Simple Idea that Does Not Work

- We have sensors that report congestion....
- Have operators read sensor and adjust themselves?

Sending 4x too much
A Simple Idea that Does Not Work

- We have sensors that report congestion.

- Have operators read sensor and adjust themselves?

Increase aggregation period up to 10 sec. If insufficient, use sampling.

- Sending 4x too much

- We have sensors that report congestion.

- Have operators read sensor and adjust themselves?
Challenge: Composite Policies

- Chaos if two operators are simultaneously responding to the same sensor

Incoming data → Coarsening Operator → Sampling Operator → Network

Sending 4x too much
Interfacing with Operators

- Incoming data
- Coarsening Operator
- Sampling Operator
- Network
- Controller

- Shrinking data by 50%
- Possible levels: [0%, 50%, 75%, 95%, ...]
- Sending 4x too much
- Go to level 75%
Experimental Setup

80 nodes on VICCI testbed at three sites (Seattle, Atlanta, and Germany)

Policy: Drop data if insufficient BW

Princeton
Degradation Keeps Latency Bounded

Bandwidth Shaping

95th percentile latency

Median Latency
Showing maximum latencies
## Programming Ease

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Lines of code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slow requests</td>
<td>5</td>
</tr>
<tr>
<td>Requests by URL</td>
<td>5</td>
</tr>
<tr>
<td>Bandwidth by node</td>
<td>15</td>
</tr>
<tr>
<td>Bad referrers</td>
<td>16</td>
</tr>
<tr>
<td>Latency and size quantiles</td>
<td>25</td>
</tr>
<tr>
<td>Success by domain</td>
<td>30</td>
</tr>
<tr>
<td>Top 10 domains by period</td>
<td>40</td>
</tr>
<tr>
<td>Big Requests</td>
<td>97</td>
</tr>
</tbody>
</table>
Conclusions and Future Work

- Useful to embed aggregation and degradation abstractions in streaming systems.
- Aggregation can be unified with storage.
- System must accommodate degradation semantics.
- Open questions:
  - How to guide users to the right degradation policy?
  - How to embed abstractions in higher-level language?